Implementación de los Módulos de Convocatoria de Artículos y Evaluación de Artículos para el Portal Web del Componente 8 del Proyecto VLIR-ESPOL Utilizando MERODE como Metodología de Análisis y J2EE como Arquitectura de Diseño

Salomón Alberto Herrera Layana
Facultad de Ingeniería en Electricidad y Computación
Escuela Superior Politécnica del Litoral
Guayaquil, Ecuador
salomon.herrera@ieee.org

María Verónica Macías Mendoza
Facultad de Ingeniería en Electricidad y Computación
Escuela Superior Politécnica del Litoral
Guayaquil, Ecuador
mmacias@fiec.espol.edu.ec

Resumen

J2EE es una plataforma de programación para desarrollar y ejecutar software de aplicaciones con arquitectura de múltiples capas distribuidas. Se basa ampliamente en componentes modulares de software ejecutándose sobre un servidor de aplicaciones. MERODE es una metodología de análisis orientada a objetos que se basa en el principio de dependencia de existencia para modelar el negocio de manera fácil y flexible, y utiliza métodos formales para asegurar la calidad del modelo resultante. El presente artículo resume la implementación de los módulos de convocatoria de artículos y evaluación de artículos para el portal web del Componente 8 del proyecto VLIR-ESPOL, basada en el uso de MERODE en la etapa de análisis y la arquitectura J2EE en la implementación. Finalmente, presentamos los resultados de las métricas tomadas durante el desarrollo del proyecto, en relación a los cambios de fondo, forma y error, demostrando que ocurren un menor número de cambios en las capas más internas que en las externas como asegura la metodología MERODE.

Palabras Claves: patrones, MERODE, J2EE, mantenimiento, análisis, diseño.

Abstract

J2EE is a Programming Architecture for developing multitier enterprise applications. It is based on software modular components running over an application server. MERODE is an Object Oriented Analysis method that use the principle of existence dependency for modeling the business in an easy and flexible way and it uses formal methods to ensure the final model quality. This paper summarizes the implementation of call for paper and evaluation papers modules for the Web portal of Component 8 VLIR-ESPOL project based in the use of MERODE during the analysis phase and the J2EE Architecture in the implementation phase. Finally, we present the results of the metrics taken during the project development in relation to changes of essence, form and error, demonstrating that occur fewer changes in the internal layers than in the external layers as it is assured by MERODE methodology.

1. Introducción

El componente 8 del proyecto VLIR-ESPOL tiene como objetivo principal el desarrollo de la capacidad de la educación e investigación para la Ingeniería de Software, Telecomunicaciones y Robótica. [6].

Entre las actividades del subcomponente de Ingeniería de Software, se encuentra el estudio, difusión y aplicación de la metodología de análisis MERODE para desarrollo de sistemas. Esta metodología ha sido utilizada en el análisis para la implementación de los módulos de Convocatoria y Evaluación de artículos técnicos para eventos. Estos módulos serían añadidos a la funcionalidad principal del portal web del Componente 8 e implementados bajo la arquitectura J2EE.

El presente artículo muestra en la primera parte una descripción de la metodología MERODE y las técnicas que utiliza en el análisis. Seguidamente, se describe la arquitectura J2EE y los patrones que se utilizaron en el diseño de los módulos. Luego se explica cómo se aplicaron los patrones de diseño al análisis de los módulos, para finalmente demostrar a través de los resultados obtenidos de las métricas que, tal como asegura la metodología MERODE, habrá menos cambios en la capa más interna que en las externas, lo que garantiza un fácil mantenimiento de los sistemas.

2. Contenido

2.1. Descripción de la metodología MERODE

MERODE es una metodología de análisis orientada a objetos diseñada en el Departamento de Ciencias Económicas Aplicadas de la Universidad Católica de Leuven, Bélgica [1]. Las siglas MERODE significan Model-based Existence-dependency Relationship Object-oriented Development (Desarrollo Orientado a Objetos basado en el modelo de las relaciones de Dependencia de Existencia) [1].

MERODE divide al sistema en 3 capas: la Capa Interna representa la fase de modelamiento del Negocio, la capa media representa la fase de modelamiento funcional y la capa externa la fase de Modelamiento de Interfaz de Usuario [1]. En la figura 1 se puede apreciar la distribución de las capas.

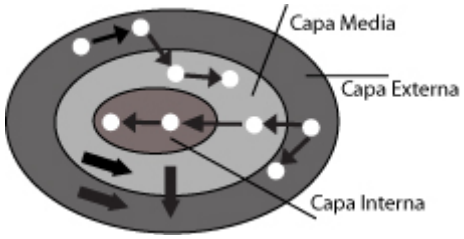

Figura 1. Representación gráfica del EDG.

Los objetos de cada capa solo pueden comunicarse entre objetos de la misma capa o con aquellos de la capa inferior inmediata, lo que conlleva a que los cambios de un objeto solo afecten a objetos de capas superiores y de la misma capa. Entonces, si colocamos los objetos que menos cambien en las capas inferiores, decrementamos el esfuerzo a realizar en futuros cambios.

Como se mencionó anteriormente la característica de dependencia de existencia es la más importante en la metodología MERODE. Esta característica está basada en la notación de la vida de un objeto, es decir, el periodo que existe entre el momento de su creación hasta el momento de su finalización, por tanto, siendo P y Q dos tipos de objetos, el principio afirma que P es dependientemente existente de Q si y solo si la vida de cada ocurrencia de p de tipo P está embebida en la vida de una y siempre la misma ocurrencia de q de tipo Q [1].

Para el caso anterior, p es llamado “objeto dependiente” y su existencia depende de la de q, y a su vez, q es llamado “objeto maestro”.

Refiriéndonos al periodo de vida de los objetos dependiente y maestro, la vida de un objeto dependiente no puede empezar antes que la del objeto maestro y así mismo, no puede terminar después que la del objeto maestro.

MERODE tiene 3 técnicas que son: El gráfico de dependencia de existencia, la tabla de eventos de objetos y finalmente las restricciones de secuencia.

A continuación explicamos brevemente cada una de sus técnicas.

2.1.1. El gráfico de la dependencia de existencia (EDG. Existence Dependency Graph) EDG. Siendo M el conjunto de clases en el Modelo del Dominio, el EDG es un ordenamiento parcial de estas clases tal que una clase nunca depende de sí misma y el gráfico es acíclico.

En la figura 2 se muestra la representación gráfica de la relación de dependencia de existencia, el tipo de objeto maestro debe ser colocado arriba del tipo de objeto dependiente y ambos conectados con una línea.

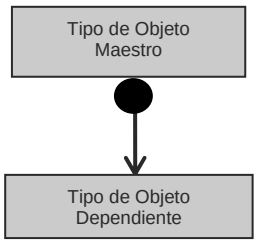

Figura 2. Representación gráfica del EDG.

2.1.2. La tabla de eventos de objetos (OET, Object-Event Table). El OET sirve para representar los aspectos dinámicos del dominio, muestra todos aquellos objetos del negocio que son afectados por cada evento del negocio y la forma en que son

afectados. Los eventos del negocio ocurren en algún instante de tiempo, no tienen duración, suceden en el mundo real y son atómicos (no pueden ser separados en sub-eventos). Para identificar los eventos del negocio debemos tener presente: que por cada objeto del negocio se tiene mínimo dos eventos del negocio, uno creador y otro finalizador, y que por cada objeto del negocio es posible tener uno o varios eventos modificadores.

Es una tabla que contiene una fila por cada tipo de evento y una columna por cada tipo de objeto. Cada celda contiene una 'C' (si representa un objeto creador), una 'E' (si representa un objeto finalizador), una 'M' (si representa un objeto modificador) o un espacio en blanco (cuando el objeto de la columna correspondiente no es afectado por el evento de la fila correspondiente). El OET sirve para representar aspectos dinámicos, mostrando que objetos del negocio son afectados por cada evento del negocio y en que forma [1], en la figura 3 se muestra gráficamente el OET.

	Obj1	Obj2
crear_obj1	C	
fin_obj1	E	
crear_obj2		C
fin_obj2		E
mod_obj1	M	
mod1_obj2		M
mod2_obj2		M
fin2_obj2		E

Figura 3. Tabla de Eventos de Objetos

2.1.3. Restricciones de secuencia y Maquina de Estado Finito. Las restricciones de secuencia son aquellas restricciones de secuencia de eventos específicas para cada tipo de objeto.

Para modelar las restricciones de secuencia, MERODE utiliza la Máquina de Estado Finito (FSM). La máquina de estado finito contiene dos ejes, el progreso hacia el estado final es representado en el eje horizontal y las partes cíclicas de la secuencia son representadas en el eje vertical. En la figura 4 podemos apreciar un diagrama de un FSM por defecto.

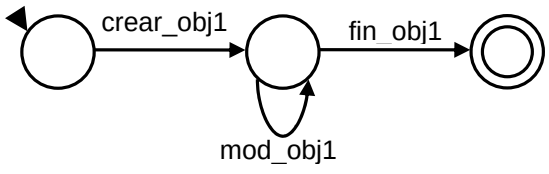

Figura 4. Ciclo de vida por defecto representado como FSM del objeto obj1

2.2. Descripción de la Arquitectura J2EE. J2EE, Java 2 Enterprise Edition o Java EE, como habíamos mencionado, es una plataforma de programación, parte de la Plataforma Java, para desarrollar y ejecutar software de aplicaciones en Lenguaje de programación Java con arquitectura de múltiples capas distribuidas, basándose ampliamente en componentes modulares de software ejecutándose sobre un servidor de aplicaciones [5].

En la figura 5 se muestra un esquema de aplicaciones multicapas. En la capa del usuario se encuentran las vistas con las que interactúan el cliente, en la capa lógica está toda la lógica del sistema y en la capa de datos están los repositorios de datos.

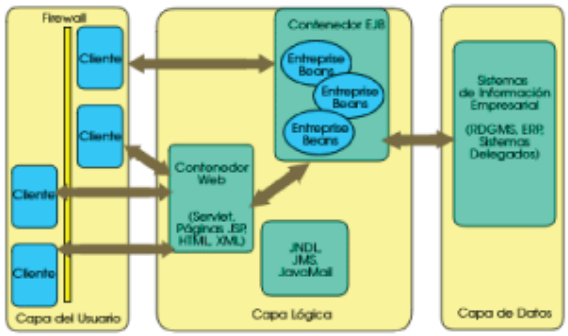

Figura 5. Aplicaciones multicapas

Para el desarrollo de este proyecto se ha tomado de referencia los patrones de diseños: Composite View y Model View Controller.

2.2.1. Composite View (Vistas compuestas). Este patrón proporciona un mecanismo para combinar modularmente, las porciones atómicas de una vista completa, las páginas son construidas uniendo código en el formato dentro de cada vista [3]. La figura 6 muestra el diagrama de la clase que representa el patrón Composite View.

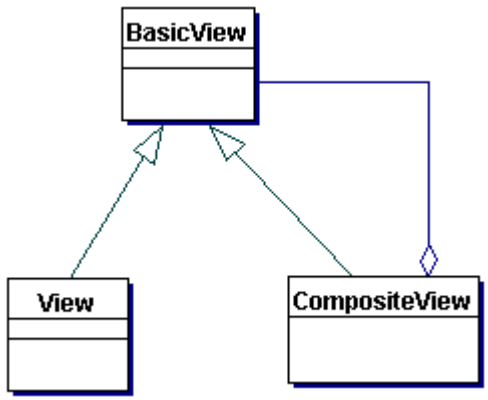

Figura 6. Diagrama de clase que representa el patrón Composite View

2.2.2. Model View Controller (Modelo Vista-Controlador) MVC. Organiza los objetos en una de las tres categorías: modelos para mantenimiento de los datos, vistas para mostrar todo o una porción de los datos y controlador para manejar los eventos que afectan las vistas o el modelo. MVC separa detalles del

diseño (persistencia, presentación, y control de los datos), disminuye la duplicación de código, centraliza el control de la aplicación y hace que los cambios o actualizaciones sean más fácilmente manejables [2].

Un diseño MVC puede centralizar el control de funcionalidades como el uso de seguridad, de sesión, y el flujo de la pantalla. MVC puede adaptarse a nuevas fuentes de datos, creando código que adapta la nueva fuente con las pantallas. Finalmente, MVC define claramente las responsabilidades de las clases que participan.

2.3. Análisis de los módulos

En esta sección analizaremos lo realizado en el análisis de los módulos que se implementaron.

2.3.1. Requerimientos funcionales. Resultaron 16 para el módulo de convocatoria de artículos y 33 para el módulo de evaluación de artículos.

2.3.2. EDG. Los objetos de los nuevos módulos debían ser añadidos al modelo inicial con el que fue diseñado el portal del Componente 8. Se necesitaron un total de 23 objetos para implementar los dos módulos; de los cuales 10 objetos son adicionales al EDG original. Actualmente, en total el portal consta de 42 objetos del negocio que conforman su EDG final.

2.3.3. OET. Debido a que es muy extensa la tabla de eventos se ha procedido a separarla por objetos para que sea visualizada de una mejor manera, como podemos apreciar en las tablas 1, 2 y 3.

Tabla 1. OET del objeto Convocatoria

	Convocatoria
cr_Convocatoria	O/C
end_Convocatoria	O/E
modificar_Convocatoria	O/M
cr_Tema	A/M
end_Tema	A/M
cr_Tema_Articulo	A/M
end_Tema_Articulo	A/M
modificar_Tema	A/M

Tabla 2. OET del objeto Tema_Articulo

	Tema_Articulo
cr_Tema_Articulo	O/C
end_Tema_Articulo	O/E

Tabla 3. OET del objeto Tema

	Tema
cr_Tema	O/C
end_Tema	O/E
cr_Tema_Articulo	A/M
end_Tema_Articulo	A/M
modificar_Tema	O/M

Las tres tablas presentadas anteriormente especifican que eventos (filas) afectan a cada uno de los objetos (columna) del modelo.

2.3.4. Máquina de estado finito del objeto Articulo. Como se muestra en la figura 7 el objeto Articulo no tiene un ciclo de vida por defecto. Este objeto representa todos aquellos artículos que recibe el grupo de investigación en cada una de las convocatorias de los eventos que organiza y a su vez, aquellos que publiquen sus investigadores por los trabajos de investigación realizados.

El objeto Articulo se crea mediante el evento cr_Articulo y su estado cambia a existe. Una vez creado, el objeto cambia al estado evaluando, cuando los organizadores de un evento asignan al menos un evaluador al artículo.

Luego, los evaluadores asignados deben realizar la revisión del artículo y llenar la evaluación.

Una vez que el comité organizador recibe al menos una evaluación de un revisor, el objeto pasa al estado aprobado cuando se acepta al artículo como ponencia de un evento, caso contrario si el comité decide no aprobarlo, el objeto pasa al estado de rechazado.

Una vez que el objeto ha sido aceptado, el autor del artículo deberá enviar la corrección en base a las sugerencias de los revisores y una vez recibida, el objeto pasa al estado corregido, y recién en este estado el autor del artículo es considerado como ponente del evento.

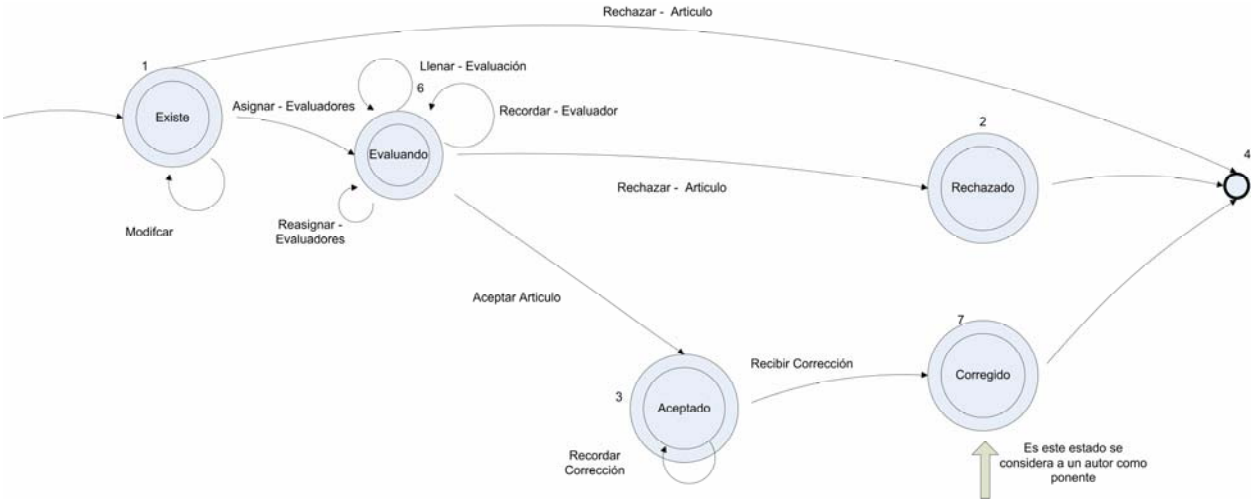

Figura 7. Diagrama FSM para el objeto Articulo

2.4. Diseño de los módulos

2.4.2. Composite View. Este patrón fue utilizado para componer las vistas resultantes que serán visibles para los usuarios.

Para ilustrar mediante código jsp (java server page) el uso del patrón, podemos apreciar en la figura 8 como se implementa la composición de vistas.

Como podemos notar a través de la sentencia include, se invoca a cada una de las partes que conforman la página principal.

```
<body>
    <div id="div_body_general">
        <div class="align_right">
            <%@include file="../../menu_general.jsp" %>
        </div>
        <%@include file="encabezado_evento.jsp" %>
        <%@include file="../../usuario_registrado.jsp"%>
        <div id="div_body_evento">
            <div id="div_izquierdo">
                <%@include file="menu_administracion.jsp" %>
            </div>
```

Figura 8. Ejemplo de aplicación del MVC en el ingreso de un pregunta en una evaluación

2.4.3. Model View Controller. A continuación se muestra la figura 9 que presenta un ejemplo de la adaptación del MVC a la implementación de los módulos al portal.

En este caso, se refiere al ingreso de una pregunta cuando se está creando una evaluación para la revisión de artículos en una convocatoria de un evento.

El usuario a través de la vista ingreso-pregunta.jsp, hace un ingreso de los datos de una pregunta y la vista hace una invocación al controlador F_Pregunta.java, quien a su vez, llama al modelo para interactuar con la clase Pregunta.java para poder acceder a la base de datos a través de la clase BaseDatos.java.finalmente para mostrar las preguntas que se han ingresado en determinada sección la clase F_Pregunta.java trabaja con la clase F_mostrar_preguntas.java, quien toma de la base de datos los registros correspondientes a través de la clase BaseDatos.java.

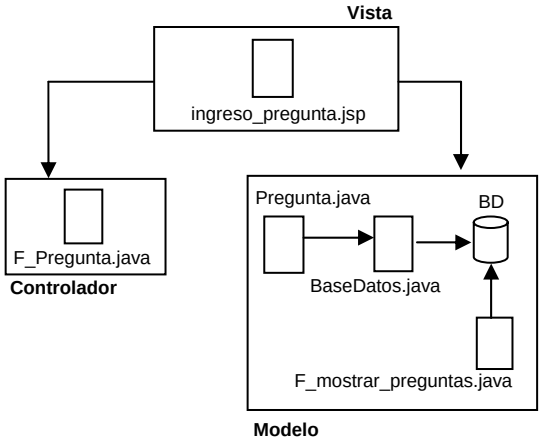

Figura 9. Ejemplo de aplicación del MVC en el ingreso de un pregunta en una evaluación

2.5. Métricas tomadas

2.5.3. Cantidad de cambios de fondo, forma y error en cada módulo. En la tabla 6 se muestra el resultado de la métrica tomada. A través de esta métrica íbamos a medir aquellos cambios de fondo, es decir, aquellos en cuanto a los requerimientos que iban a afectar el EDG. Luego, todos aquellos cambios de visualización, colores, posición, mensajes en las pantallas, estos serían los cambios de forma. Finalmente, los cambios de error, los cambios por errores internos del sistema al momento de la implementación.

Con esta métrica queríamos demostrar que al aplicar la metodología MERODE para el análisis de sistemas, íbamos a encontrar que ocurrieron un menor número de cambios en la capa más interna (cambios de fondo) que en la capa más externa (cambios de forma y error).

Tabla 4. Tabla de cambios de fondo, forma y error en los componentes

Componente	Cambios		
	Fondo	Forma	Error
Convocatoria	0	23	12
Evaluación	2	28	14
Total de Cambios	2	51	26
Porcentaje (%)	1.27%	64.56%	32.91%

2.5.1. Tiempo real utilizado para el desarrollo de cada módulo (medido en días laborables). En la tabla 4 se muestra el resultado de la métrica tomada. El Tiempo A se refiera al tiempo real menos el tiempo estimado y el porcentaje es el tiempo A en relación al tiempo estimado.

Tabla 5. Tiempos utilizados en el desarrollo de los módulos

Componente	T. estimado	T. real	Tiempo A	%
Convocatoria	40	73	13	83%
Evaluación	40	49	9	23%

2.5.2. Tiempo real vs. Tiempo estimado inicial de la codificación del proyecto en general (medido en meses). En la tabla 5 se muestra el resultado de la métrica tomada. El Tiempo A se refiera al tiempo real menos el tiempo estimado y el porcentaje es el tiempo A respecto al tiempo estimado.

Tabla 6. Tiempos utilizados en el desarrollo de todo el proyecto

	T. estimado	T.real	Tiempo A	%
Proyecto General	4	6.1	2.1	52.5%

3. Conclusiones y recomendaciones

Se explicó las ventajas de adoptar la metodología MERODE en el análisis de sistemas, para finalmente comprobar que el mantenimiento resulta muy fácil de administrar.

Se utilizó MERODE como metodología de análisis y resultó muy práctico aplicar el patrón MVC que propone la arquitectura J2EE para la implementación de la funcionalidad, puesto que para ambos casos trabajan al sistema en capas estructuradas.

Se demostró que el porcentaje de cambios que afectan las capas más internas en MERODE fue mucho menor que aquellos de las capas superiores, donde el porcentaje de cambios de forma y error fueron muy altos.

Se comprobó que MERODE ofrece flexibilidad al identificar en qué parte del código deberían hacerse los cambios reduciendo el costo de mantenimiento del sistema.

La primera dificultad fue adaptar el diseño de los módulos al diseño general del portal, lo que ocasionó que el tiempo en desarrollar el módulo de Convocatoria de Artículos sea mucho mayor.

Se recomienda que al añadir funcionalidad al portal se considere el uso de patrones de la arquitectura J2EE en el análisis realizado con MERODE pues resultó bastante semejante la utilización del patrón MVC a la arquitectura de la metodología.

4. Agradecimientos

Este artículo fue realizado gracias al apoyo del Subcomponente de Ingeniería de Software del Componente 8 del Proyecto VLIR-ESPOL.

5. Referencias

[1] Snoeck, M., Dedene, G., Verhelst, M., Depuydt, A-M., Object-Oriented Enterprise Modelling with MERODE, Leuven University Press, 1999.

[2] Página de referencia del MVC, http://www.enode.com/x/markup/tutorial/mvc.html

[3] Core J2EE Patterns Best Practices and Design Strategies, http://www.corej2eepatterns.com/Patterns2ndEd/CompositeView.htm

[4] Core J2EE Patterns Best Practices and Design Strategies, http://www.corej2eepatterns.com/Patterns2ndEd/DataAccessObject.htm

[5] Java 2 Platform, Enterprise Edition (J2EE) Overview, http://java.sun.com/j2ee/overview.html

[6] Proyecto de investigación VLIR-ESPOL, http://www.cicyt.espol.edu.ec/vlir/index.htm